**DISTRIBUTED SYSTEMS**

**Laboratory Assignment 1**

**ONLINE ENERGY UTILITY PLATFORM**

Student: Bucur Alexandra

Group: 30441

2022-2023

Contents

[Conceptual architecture of the online platform. 3](#_Toc118888930)

[Overview 3](#_Toc118888931)

[Technologies 3](#_Toc118888932)

[Design 4](#_Toc118888933)

[DataBase design 4](#_Toc118888934)

[Backend design 4](#_Toc118888935)

[Class diagram 4](#_Toc118888936)

[Package design 5](#_Toc118888937)

[Deployment 7](#_Toc118888938)

[Deployment files 7](#_Toc118888939)

# Conceptual architecture of the online platform.

## Overview

The Energy Utility platform is an online platform designed and implemented to manage users, their associated smart energy metering devices, and the monitored data from each device.

The system can be accessed by two types of users after a login process: administrator (manager), and clients. The administrator can perform CRUD (Create-Read-Update-Delete) operations on user accounts (defined by ID, name, role: admin/client), registered smart energy metering devices (defined by ID, description, address, maximum hourly energy consumption), and on the mapping of users to devices (each user can own one or more smart devices in different locations).

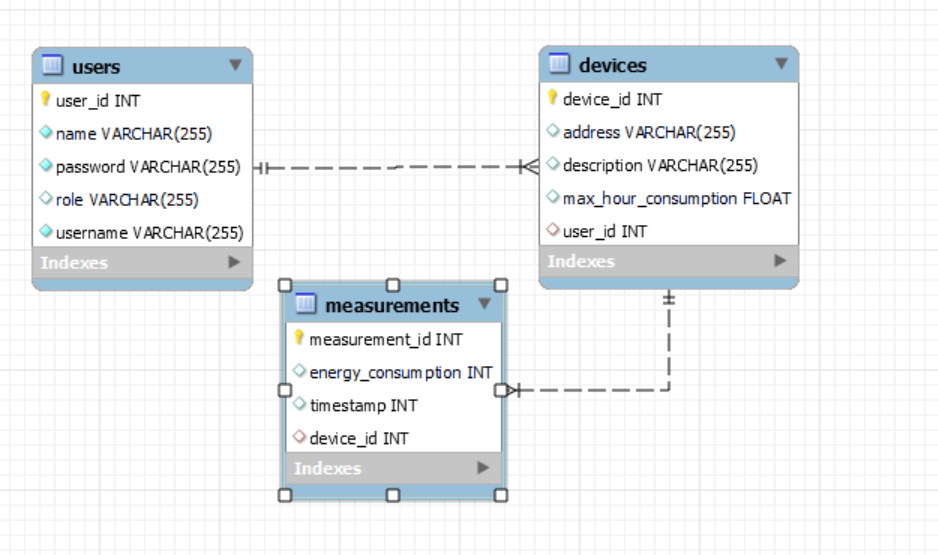
## Technologies

Energy Utility Platform is an online platform build using RESTful services and JavaScript-based client applications. It is composed of the following modules:

* Energy Database: a **Posgres** database that stores all the information about:
  + **Users**: account details (name, password etc..) and role (administrator/client)
  + **Devices:** that can be associated to clients that can be individually monitored by a sensor. A user can have multiple devices, but a device only one user.
  + **Measurements**: data that is stored regarding monitored devices in touples of format <timestamp, consumption>
* Backend: built services in **Java Spring Boot** framework. The code is organized by a layered architecture. It manages the calls to the database and give access to it. It has the associated models: User, Device and Measurements.
* Frontend: an online platform based on **Angular** framework that helps the user to use the application in order to:
  + Admin:
    - * perform CRUD operations on Devices
      * perform CRUD operations on Users
      * perform user-device mappings
  + User
    - * visualize the devices
      * see charts based on the measurements
* Deployment: **Docker**

# Design

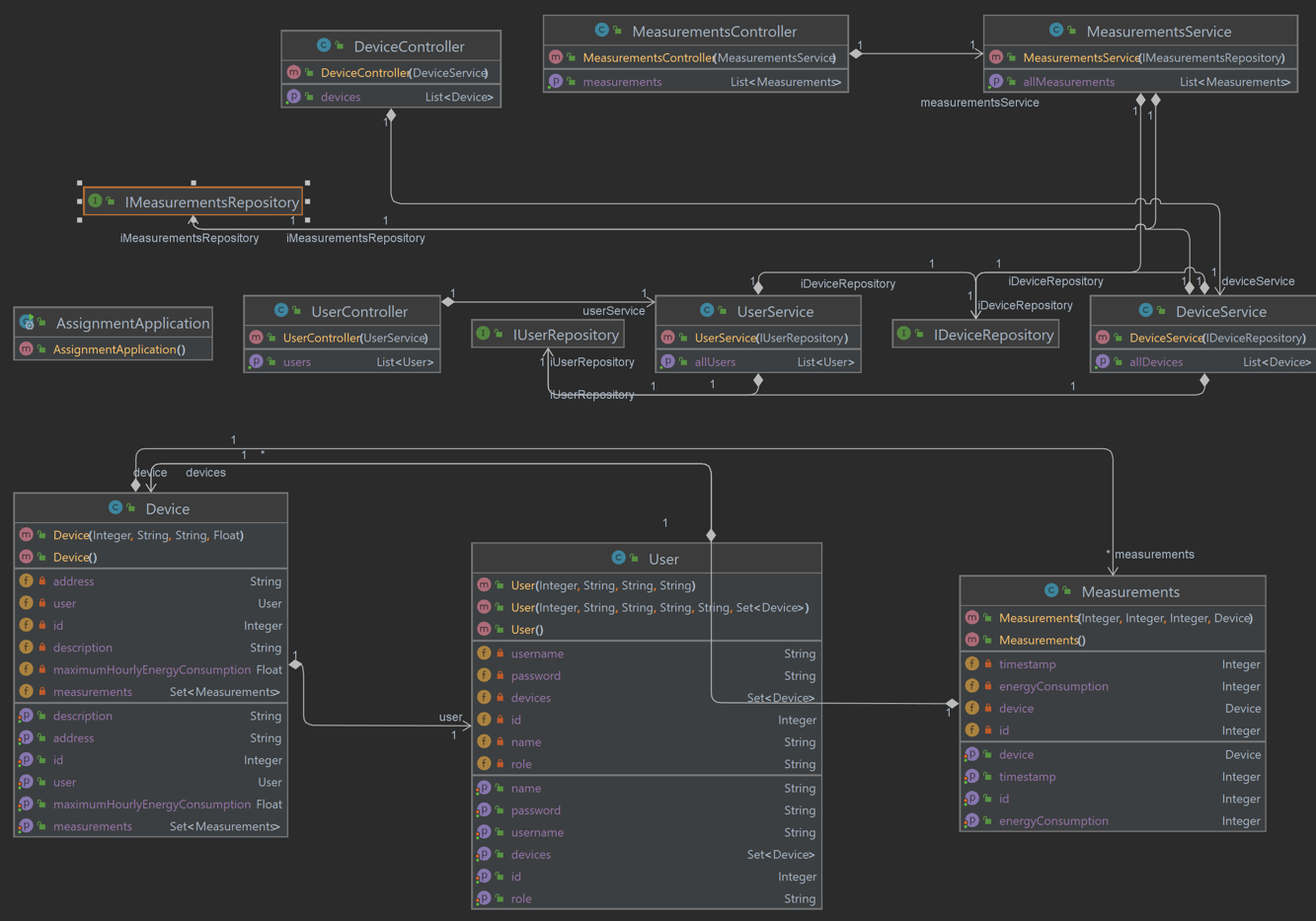
## DataBase design



## Backend design

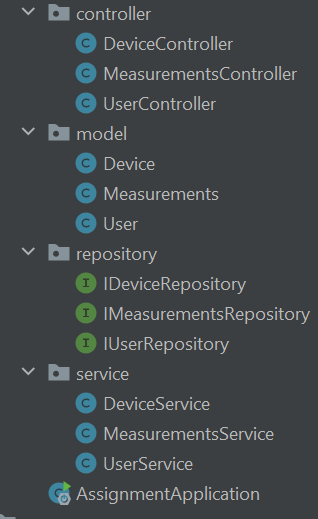
### Class diagram

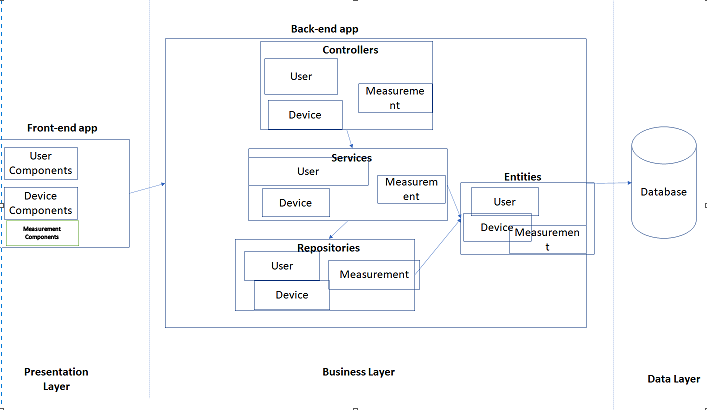
The backend was designed using layered architecture



### Package design

* **Controllers** :
  + - * + top level
        + API calls are performed, thus helping with the communication with the frontend part
        + Represents the presentation layer
* **Services:**
  + - * + Contains methods implementations of the CRUD operations that are called from the controller
        + Performs validations of data
* **Repositories:**
  + - * + layer between models and services
        + usually contains interfaces
        + extends JPARepository, thus only contains queries needed for the operations of data
* **Models:**
  + - * + Represents a 1:1 mapping with the database
        + Has several objects that each represent a table from the DB



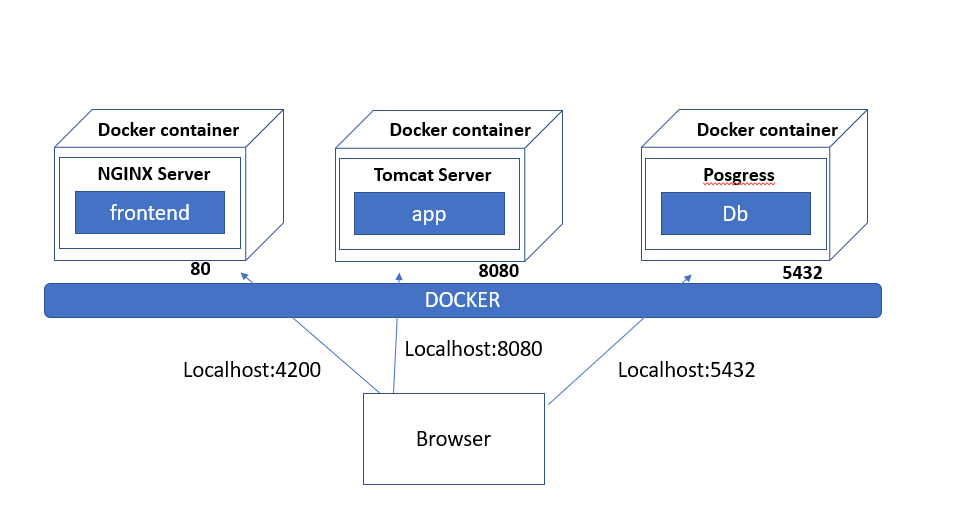


# Deployment

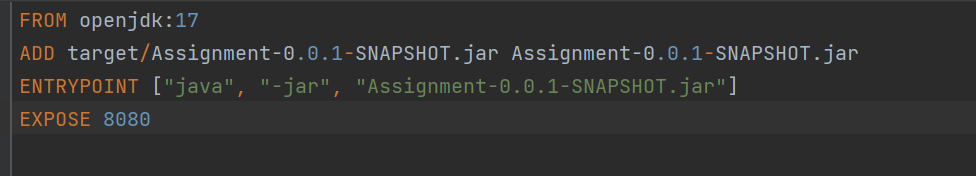
The deployment is done on Docker using different containers for the different part of application, so there are 3 containers: backend, frontend and database. Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker’s methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production. Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allows you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host. You can easily share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

## Deployment files

#### Diagram



### Backend



Compose

version: '2'  
  
services:  
 app:  
 image: 'spring\_back\_end'  
 ports:  
 - "8080:8080"  
 build:  
 context: .  
 container\_name: app  
 depends\_on:  
 - db  
 environment:  
 - SPRING\_DATASOURCE\_URL=jdbc:postgresql://db:5432/compose-postgres  
 - SPRING\_DATASOURCE\_USERNAME=compose-postgres  
 - SPRING\_DATASOURCE\_PASSWORD=compose-postgres  
 - SPRING\_JPA\_HIBERNATE\_DDL\_AUTO=update  
  
 frontend:  
 image: frontend  
 ports:  
 - "4200:80"  
 depends\_on:  
 - app  
 container\_name: frontend  
  
 db:  
 image: 'postgres:13.1-alpine'  
 container\_name: db  
 ports:  
 - "5432:5432"  
 environment:  
 - POSTGRES\_USER=compose-postgres  
 - POSTGRES\_PASSWORD=compose-postgres  
 - POSTGRES\_DB=compose-postgres

### Frontend

